Fluid Directed Rigid Body Control using Deep Reinforcement Learning

PINGCHUAN MA, Nankai University, China
YUNSHENG TIAN*, Nankai University, China
ZHERONG PAN, University of North Carolina at Chapel Hill, USA
BO REN†, Nankai University, China
DINESH MANOCHA, University of Maryland at College Park, USA

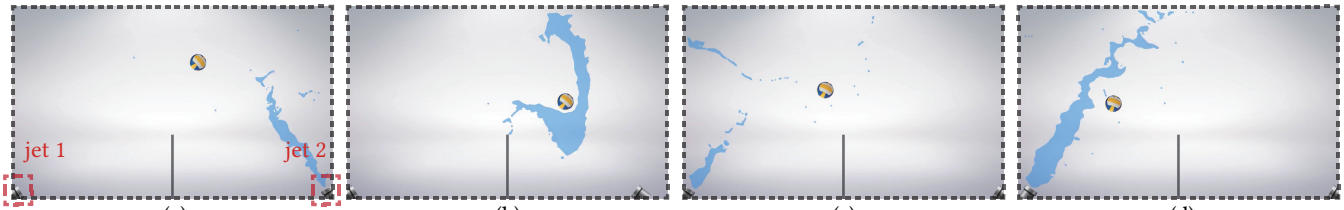

Fig. 1. Our new algorithm based on deep reinforcement learning can be used to control 2D coupled fluid/rigid animations. We use fluid jets (red blocks in (a)) to control a rigid body to exhibit desired behaviors. In this example, we use two fluid jets to play a cooperative ball game, where the goal is to push the ball back and forth. In these frames, the ball first flies to the right (a) and the right jet shoots the ball back (b). Next, the ball flies to the left (c), the left jet shoots it back (d). We use DRL to compute appropriate ghost forces at the fluid-solid boundary to control the ball's trajectory.

We present a learning-based method to control a coupled 2D system involving both fluid and rigid bodies. Our approach is used to modify the fluid/rigid simulator's behavior by applying control forces only at the simulation domain boundaries. The rest of the domain, corresponding to the interior, is governed by the Navier-Stokes equation for fluids and Newton-Euler's equation for the rigid bodies. We represent our controller using a general neural-net, which is trained using deep reinforcement learning. Our formulation decomposes a control task into two stages: a precomputation training stage and an online generation stage. We utilize various fluid properties, e.g., the liquid's velocity field or the smoke's density field, to enhance the controller's performance. We set up our evaluation benchmark by letting controller drive fluid jets move on the domain boundary and allowing them to shoot fluids towards a rigid body to accomplish a set of challenging 2D tasks such as keeping a rigid body balanced, playing a two-player ping-pong game, and driving a rigid body to sequentially hit specified points on the wall. In practice, our approach can generate physically plausible animations.

CCS Concepts: • **Computing methodologies** → **Physical simulation**;

Additional Key Words and Phrases: fluid/rigid coupling, optimal control, reinforcement learning

*joint first author
†corresponding author

Authors' addresses: Pingchuan Ma, Nankai University, 94 Weijin Rd, Tianjin, 300071, China, pika7ma@gmail.com; Yunsheng Tian, Nankai University, 94 Weijin Rd, Tianjin, 300071, China, knoxtian@gmail.com; Zherong Pan, University of North Carolina at Chapel Hill, Sitterson Hall, Chapel Hill, NC, 27514, USA, zherong@cs.unc.edu; Bo Ren, Nankai University, 94 Weijin Rd, Tianjin, 300071, China, rb@nankai.edu.cn; Dinesh Manocha, University of Maryland at College Park, A.V. Williams Building, 8223 Paint Branch Drive, College Park, MD, 20742, USA, dm@cs.unc.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
0730-0301/2018/8-ART96 $15.00
https://doi.org/10.1145/3197517.3201334

ACM Reference Format:
Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. 2018. Fluid Directed Rigid Body Control using Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 4, Article 96 (August 2018), 11 pages. https://doi.org/10.1145/3197517.3201334

1 INTRODUCTION

Fluid animation has been widely studied in computer graphics and related areas. Over the years, different techniques have been proposed to achieve visual plausibility on desktop machines. As a result, fluid simulators are used as standard tools in animation systems, e.g., Houdini and Naiad. Besides generating visually plausible animations, an additional requirement is for an artist to edit or control the fluid behaviors in a goal-directed or intuitive manner. Numerous methods, such as [Fattal and Lischinski 2004; McNamara et al. 2004; Pan et al. 2013; Pan and Manocha 2017; Shi and Yu 2005; Thuerey 2016; Treuille et al. 2003], have been proposed to this end.

The current control schemes do not account for the interactions between the fluid and other physical objects in the scene, such as solid boundaries or free-flying rigid objects. Ignoring these objects is reasonable for keyframe-based control applications where the goal is to deform the fluid into a target shape. However, in many other scenarios, the resulting simulation and control are based on fluid/rigid-body interactions. Typical applications of such scenarios include controlling a floating boat in a movie and controlling a volleyball using water jets in an AI game. In these scenarios, control is facilitated by applying forces on the fluid body, and the fluid body, in turn, influences the rigid body by applying forces and torques. Many prior algorithms [Fattal and Lischinski 2004; Shi and Yu 2005] apply virtual artificial control forces, or the so-called ghost forces, in the simulation domain to achieve controlling goals. Instead, we consider a much more challenging problem of boundary-condition-induced control, i.e. the controller only influences the fluid body at the simulation domain boundaries through physical actions

(e.g., using steerable water jets). Therefore, the controller serves as providing variable boundary conditions, exactly preserving the natural physical governing law over the simulation domain with no artificial force terms added.

It is rather difficult to extend previous fluid control methods to solve our problem of controlling fluid/rigid systems. Some methods [McNamara et al. 2004; Pan and Manocha 2017; Treuille et al. 2003] formulate the fluid control problem as a continuous trajectory optimization, which can be solved using gradient-based optimization. However, with fluid/rigid interactions, the resulting control problem becomes a non-smooth optimization that cannot be solved continuously. The non-smoothness arises from the frequent contact point changes between fluid and rigid body interfaces. In addition to trajectory optimization, [Fattal and Lischinski 2004; Shi and Yu 2005] describe simple methods to control the fluid by matching the fluid shape with a target shape. However, these techniques assume that the target shape of the fluid is known as a prior. Unfortunately, in our problem, the concept of targets is only well-defined for rigid bodies, but not for fluid bodies. On the other hand, it is difficult to extend techniques for low-DOF rigid body control [Popović et al. 2003, 2000; Twigg and James 2007, 2008] to handle such coupled cases. This is because the controller needs to account for the high-dimensional state of the fluid body to achieve the indirect control of the rigid body, resulting in a high-DOF control problem.

1.1 Main Results

Inspired by recent advances in deep reinforcement learning (DRL) for animation problems [Liu and Hodgins 2017; Peng et al. 2017; Won et al. 2017], we present a novel learning-based approach to control coupled fluid/rigid systems in 2D domains. DRL has been shown to exhibit good performance in terms of controlling non-smooth dynamic systems, such as contact-rich character locomotions. In our formulation, the control task is decomposed into two stages: the training stage and the generation stage. During the training stage, we parameterize the controller using a two-block neural-net, where the first block extracts the low-dimensional features from high-dimensional fluid states and the second block maps the features to control inputs. We optimize the first block using an auto-encoder and the second block using a DRL algorithm [Schulman et al. 2015a]. Our proposed network design allows the simultaneous training of the two network blocks to reduce the time cost in the training stage. After the training, coupled fluid/rigid animations can be generated at realtime with the small overhead of evaluating the neural-net.

Our approach achieves efficient and effective control that preserves exact natural physical dynamics on a 2D coupled system. We have evaluated our method on a set of 2D benchmarks as illustrated in Figure 2. We first achieve effective control in simple tasks such as keeping a single rigid ball at a fixed altitude by counteracting gravitational forces using liquid jets. We have also evaluated the performance on more complex tasks involving two controllers, where the goal is to play a ping-pong or volleyball game (frames are illustrated in Figure 1). In addition to controllers that achieve a single goal, our method can also train a multi-target controller to drive a ball to hit different points on a ceiling at specified time instances. Finally, we extend our method to 3D scenarios, where we control

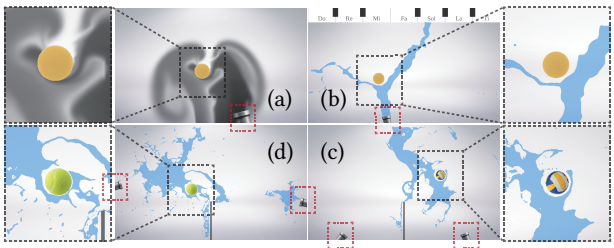

Fig. 2. We show different control tasks generated by our realtime algorithm. (a): We keep the ball in the center of screen by counteracting gravity. (b): We drive the ball to hit different keys (blue blocks) on the ceiling. (c,d): We run a two-player ball game, where the two players can be either cooperative (c) or competitive (d). All these control tasks are realized using fluid jets located in the boundary of the simulation domain (red blocks).

a rigid ball floating on water simulated using the shallow water equation. Training time ranges from a few hours to about one day, from the simplest to the most complex benchmarks. These results show that our algorithm can provide robust control over different rigid body shapes, control objectives, and fluid-related features.

The rest of the paper is organized as follows. We first review related works on fluid simulation, fluid control, and learning-based control in Section 2. Then we formulate the coupled fluid control problems in the DRL setting in Section 3. Our approach to train the controller is described in Section 4 and results are highlighted in Section 5.

2 RELATED WORK

In this section, we review previous works on fluid simulation, fluid control, and learning-based control.

2.1 Fluid Simulation

Fluid Simulation has been extensively studied in computer graphics and different techniques have been proposed. Fluid simulators can be categorized according to the discretization methods, which include grid-based methods [Enright et al. 2002], particle-based methods [Becker and Teschner 2007], and hybrid methods [Losasso et al. 2008; Zhu and Bridson 2005]. Simulators can also be categorized according to the type of underlying integrators, which include explicit methods [Becker and Teschner 2007], time-split methods [Zhu and Bridson 2005], and fully implicit methods [Mullen et al. 2009]. In addition, current fluid simulation systems can model general scenarios involving multiple physics models using two-way coupling. Two-way coupling methods have been developed for grid-based simulators [Batty et al. 2007; Robinson-Mosher et al. 2008] and particle-based simulators [Akinci et al. 2012; Becker et al. 2009]. Grid-based simulators perform more computations during each timestep, but can take larger timestep sizes. On the other hand, particle-based simulators are faster during each timestep, but work under small timesteps.

2.2 Fluid Control

Fluid Control algorithms are developed on top of fluid simulators and provide additional capability so that the artists can direct the fluid animations. Different fluid control methods can be classified based on the user-interfaces. Many methods [McNamara et al. 2004;

Pan and Manocha 2017; Shi and Yu 2005; Treuille et al. 2003] are keyframe-based, where the user provides a set of keyframes and the fluid tends to form similar shapes at the specific time instances. Some of these methods rely on a specific kind of fluid simulator. For example, [McNamara et al. 2004; Pan and Manocha 2017; Treuille et al. 2003] assume that the entire simulation process is a function whose gradient information can be computed, which is computationally costly. Other methods, such as [Fattal and Lischinski 2004; Shi and Yu 2005], can work with any fluid simulators. Besides keyframe-based methods, there are other techniques [Bonev et al. 2017; Raveendran et al. 2014; Thuerey 2016] that synthesize fluid animations by interpolating existing animation data. These methods do not use physically-based simulators. In practice, they are faster, but the results may not be physically correct. Contrary to our method, all these methods control a non-coupling system with only fluid body and solid boundaries, and no other dynamic objects within the simulation domain are considered. In contrast, our DRL-based controller can also work with any fluid simulator because our method treats the simulator as a black-box.

2.3 Learning-based Control

Learning-based Control methods differ from fluid control methods in that the controller is parameterized and these parameters are computed as part of a preprocess. These parameters are usually determined by optimizing the controller's performance on a set of typical control tasks. Such techniques have been used to control low-DOF dynamic systems, including articulated characters [Wang et al. 2012; Yin et al. 2007]. Some of the earlier works were limited to controllers with simple parameterization and optimized using sampling-based methods [Hansen and Ostermeier 2001]. However, prior sampling-based methods do not scale to high-DOF dynamic systems or high-DOF controller parametrizations such as controllers represented using deep neural-nets. Recent developments in DRL methods [Levine et al. 2016; Schulman et al. 2015a] have resulted in new methods to optimize the controllers, which allow us to train deep neural-nets that map directly from high-DOF observations of the environment, such as the entire fluid velocity field, to the output of the controller. Although DRL methods have been used to control articulated characters [Liu and Hodgins 2017; Peng et al. 2017; Won et al. 2017], our approach is motivated by these developments and we present a DRL method to control coupled fluid/rigid systems.

3 PROBLEM FORMULATION

In this section, we give a brief overview of the mathematical modeling of a coupled fluid/rigid system. Next, we introduce the Markov Decision Process (MDP) used by DRL to formulate the control problem.

3.1 Coupled Fluid/Rigid System

As illustrated in Figure 3, a coupled fluid/rigid system models a combined domain of fluid body Ω_f and rigid body Ω_r. Within Ω_f the fluid body is governed by the Navier-Stokes equation, while in Ω_r the rigid body is governed by Newton-Euler's equation, as follows:

$$
\begin{aligned}
&\forall \mathbf{x} \in \Omega_f: \quad \begin{cases} \dot{\mathbf{u}} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \mathbf{f} \quad \nabla\cdot\mathbf{u} = \mathbf{0} \\ \dot{\rho} + (\mathbf{u}\cdot\nabla)\rho = 0 \end{cases} \\
&\forall \mathbf{x} \in \Omega_r: \quad \begin{cases} \mathbf{u} = \mathbf{v} + \omega \times (\mathbf{x} - \mathbf{o}) \\ \mathbf{M}\begin{pmatrix} \dot{\mathbf{v}} \\ \dot{\omega} \end{pmatrix} + \begin{pmatrix} \int_{\partial\Omega_r} \mathbf{p}\, d\mathbf{s} \\ \omega \times \mathbf{I}\omega + \int_{\partial\Omega_r} \mathbf{p}(\mathbf{x}-\mathbf{c}) \times d\mathbf{s} \end{pmatrix} = 0 \\ \begin{pmatrix} \dot{\mathbf{c}} \\ \dot{\mathbf{R}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ [\omega]\mathbf{R} \end{pmatrix} \end{cases} \\
&\forall \mathbf{x} \in \Omega_f \cup \Omega_r: \quad \mathbf{u} \in C^0. \qquad (1)
\end{aligned}
$$

In the fluid region Ω_f, we denote $\mathbf{u}$ as its velocity field, p as its scalar pressure field, and $\mathbf{f}$ as the external force field. All these vectors are 2D vectors in our scenarios. We denote ρ as the fluid's density field of smoke or the level-set field of liquid, which is passively advected by $\mathbf{u}$. In the rigid region Ω_r, we denote $\mathbf{M}$ as its generalized mass-matrix, $\mathbf{v}$ as its linear velocity, and ω as its angular velocity. We denote $\mathbf{c}, \mathbf{R}$ as the rigid body's center of mass and global orientation, respectively. Finally, we enforce the boundary condition between fluid and rigid bodies by making $\mathbf{u}$ C^0-continuous everywhere in $\Omega_f \cup \Omega_r$.

In our work, the governing equations are time-integrated using the method proposed in [Robinson-Mosher et al. 2008], which is a grid-based fluid solver. However, when the fluid is a liquid, we find it important that the liquid only applies pushing forces on the rigid body. This is achieved by enforcing a positive pressure in grid cells neighboring the rigid body. We achieve this by using the wall-separating boundary condition proposed in [Batty et al. 2007]. The simulation complexity is linear in the number variables or the number of grid-cells, i.e. $O(n^2)$ in 2D cases, where n is the number of grid cells in each spatial dimension. Typically, n is around 10^2 so that the number of grid cells is about 10^4. Solving for such a large number of unknowns is the major computational bottleneck of our control problem. Therefore, we use an adaptive grid [Zhu et al. 2013] to reduce the computational overhead. Specifically, we simulate fluid using finer grids in a small subset $\Omega_f^{fine} \subset \Omega_f$ surrounding the rigid body.

Given all the variables in Equation 1, only $\mathbf{u}, \rho, \mathbf{f}, \mathbf{v}, \omega, \mathbf{c}$, and $\mathbf{R}$ are independent variables, and $\mathbf{p}$ is a dependent variable. We denote $\mathbf{f}$ in Ω_f as a control variable, which can be directly modified by the controller. All other variables are denoted as state variables, which are governed by physical laws and cannot be modified directly. In many cases, we further constraint the control variable $\mathbf{f}$ by enforcing $\mathbf{f}$ to be non-zero within only a small subset of Ω_f. Considering the scene in Figure 5, to control the rigid body using water jets, $\mathbf{f}$ can only be non-zero within the small jet source region near the domain boundary. The specific

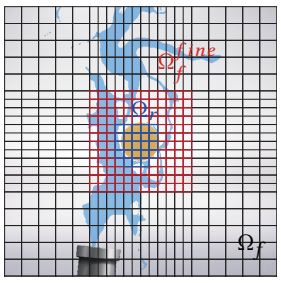

Fig. 3. We use an adaptive grid to discretize the simulation domain where the highest resolution is used around the rigid body. We illustrate different subdomains in our coupled fluid/rigid simulation: the fluid domain Ω_f (black grid region), the rigid domain Ω_r (blue grid region), and the fine domain Ω_f^{fine} (red grid region).

cases of $\mathbf{f}$ is example-dependent,
and we introduce an additional state variable $\mathbf{I}$ representing example-dependent information, e.g. the water jet location, and express the external force field as a function $\mathbf{f}(\mathbf{I})$ as illustrated in Figure 4. The final state vector is $\mathbf{S} = (\mathbf{u}\ \rho\ \mathbf{v}\ \omega\ \mathbf{c}\ \mathbf{R}\ \mathbf{I})$. Note that since the dimension of controllable variable $\mathbf{f}$ is much lower than the entire state variable $\mathbf{S}$, i.e. $|\mathbf{f}| \ll |\mathbf{S}|$, our control problem is highly under-actuated (see, for example, [Fantoni and Lozano 2001]).

3.2 Control Problem Formulation

Based on the above dynamic system, we can formulate our control problem using MDP. MDP formulation is more general then previous fluid control formulations such as trajectory optimization [Treuille et al. 2003] or shape matching [Shi and Yu 2005], which automatically takes non-smooth problems into consideration. Under this setting, the coupled fluid/rigid simulator is represented as a function $\mathcal{F}$ that brings state $\mathbf{S}_i$ to $\mathbf{S}_{i+1}$. $\mathcal{F}$ consists of two sub-equations: $(\mathbf{u}\ \rho\ \mathbf{v}\ \omega\ \mathbf{c}\ \mathbf{R})_{i+1} = \mathcal{F}_{f,r}((\mathbf{u}\ \rho\ \mathbf{v}\ \omega\ \mathbf{c}\ \mathbf{R})_i, \mathbf{f}(\mathbf{I}_i))$ $\mathbf{I}_{i+1} = \mathcal{F}_{\mathbf{I}}(\mathbf{I}_i, \mathbf{a}_i)$, where $\mathcal{F}_{f,r}$ corresponds to Equation 1. $\mathcal{F}_{\mathbf{I}}$ is the evolution function for state variable $\mathbf{I}$, and is example-dependent. For example, to control a rigid ball using a water jet, $\mathcal{F}_{\mathbf{I}}$ specifies how the water jet moves. Note that this dynamic system only takes $\mathbf{a}_i$ as input, which is called "action" in MDP terminology. The control action is generated by a so-called stochastic control policy $\pi(\mathbf{a}|\mathbf{S}, \boldsymbol{\theta})$, which is the probabilistic distribution function over the possible action set $\{\mathbf{a}\}$, given the state $\mathbf{S}$. The control policy's performance can be maximized by optimizing its parameters, $\boldsymbol{\theta}$. In our method, $\pi(\mathbf{a}|\mathbf{S}, \boldsymbol{\theta})$ is represented as a neural network and $\boldsymbol{\theta}$ are the weights of the neurons. Representing our controller using a probabilistic function instead of a deterministic function is the key to tackling non-smoothness in DRL. Finally, in order to formulate the control objectives, we assume that, on reaching the state $\mathbf{S}_{i+1}$ using control action $\mathbf{a}_i$, the controller receives a reward $r(\mathbf{S}_i, \mathbf{S}_{i+1}, \mathbf{a}_i)$. Our goal is to maximize the weighted sum of reward over all timesteps. Finally, the control problem can be formulated as an optimization problem:

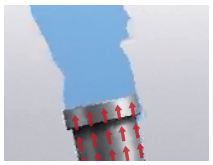

Fig. 4. A water jet can be modelled by a wind force (red arrows) directing at the spout of the water jet.

$$\underset{\boldsymbol{\theta}}{\mathbf{argmax}} \quad \mathbb{E}_{\mathbf{S}_0, \mathbf{a}_0, \cdots \sim \pi(\boldsymbol{\theta})}\left[\sum_{i=0}^{\infty} \gamma^i r(\mathbf{S}_i, \mathbf{S}_{i+1}, \mathbf{a}_i)\right], \tag{2}$$

where $\gamma < 1$ is a discount factor. The expectation $\mathbb{E}_{\mathbf{S}_0, \mathbf{a}_0, \cdots \sim \pi}[\bullet]$ is calculated over the set of possible state-action trajectories $\{\mathbf{S}_0, \mathbf{a}_0, \cdots\}$ under policy π. Note that it is necessary for the reward function to be a function of both state $\mathbf{S}$ and control action $\mathbf{a}$. For example, in keyframe-based control framework [Treuille et al. 2003], the $\mathbf{S}$-related term models the similarity between keyframes and the $\mathbf{a}$-related term models the control force regularization. Therefore, previous fluid control methods can be considered as a special case of our MDP formulation.

4 CONTROLLER DESIGN AND OPTIMIZATION

In this section, we present the details of our neural-net controller parametrization and DRL optimization algorithm. Our control pipeline is illustrated in Figure 5. As mentioned in Section 1, the two main challenges are the high-dimensionality of fluid state space $\mathbf{S}$ and the non-smoothness of the controller optimization problem. In our method, the high-dimensional $\mathbf{S}$ is reduced using a convolutional autoencoder (Section 4.1) and the non-smoothness is addressed using a stochastic policy gradient method (Section 4.2).

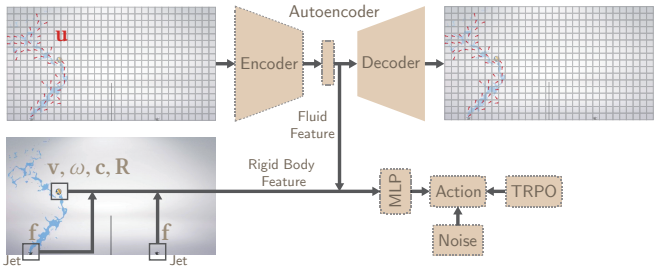

Fig. 5. We illustrate the state variables $\mathbf{u}$, $\mathbf{v}$, ω, $\mathbf{c}$, and $\mathbf{R}$. We illustrate the external force $\mathbf{f}$, which is non-zero only in the source regions (the water jet in the black box). We first extract the fluid's velocity field using a convolutional autoencoder (Section 4.1). The encoded features and other rigid features are combined and then feed into the multilayer perceptron (MLP) to get the action. The MLP and the encoder are trained using a DRL algorithm [Schulman et al. 2015a] (Section 4.2), which is marked with dashed lines.

4.1 Controller Parametrization

In this section, we define our neural-net controller $\pi(\mathbf{a}|\mathbf{S}, \boldsymbol{\theta})$, which maps from the state vector $\mathbf{S}$ to the low-dimensional action $\mathbf{a}$. In previous works [Peng et al. 2017; Won et al. 2017] for controlling articulated bodies, $\mathbf{S}$ is low-dimensional and π is parametrized using a multilayer perceptron (MLP). However, using MLP to directly map high-dimensional fluid-related state vectors $\mathbf{u}$, ρ will result in too many parameters $\boldsymbol{\theta}$ to be optimized, which in-turn requires a large amount of data to fit $\boldsymbol{\theta}$. For example, an MLP for a small 50×50 grid involves 8×10^4 parameters for the first layer with 32 neurons. Indeed, it has been shown in [Tompson et al. 2016] that the fluid's velocity field is internally correlated so that most of the dimensions can be redundant. Therefore, we choose to extract a low-dimensional feature vector from the high-dimensional velocity field $\mathbf{u}$ using a convolutional autoencoder [Masci et al. 2011; Zeiler et al. 2010]. This architecture uses less than 5×10^4 parameters to parametrize the entire controller for a 2D grid of arbitrarily high resolution.

Convolutional autoencoders belong to the category of unsupervised feature learning algorithms. Unsupervised learning is preferred in our case because labelling the velocity field is not a trivial task. We also compare the autoencoder with supervised learning, where the labels are the external forces and torques on the rigid body. However, such labels only focus on the rigid body and ignore fluid regions far away from the rigid body. Therefore, features extracted using unsupervised learning cover more comprehensive information, which results in better controller performance.

Autoencoders can be represented by a pair of encoder $\mathcal{E}(\bullet, \boldsymbol{\theta}_{\mathcal{E}})$, which maps from a high-dimensional input to a low-dimensional feature vector, and decoder $\mathcal{D}(\bullet, \boldsymbol{\theta}_{\mathcal{D}})$, which maps from the feature vector back to the high-dimensional input. The encoder and decoder are both represented by convolutional neural-nets (CNN) with a

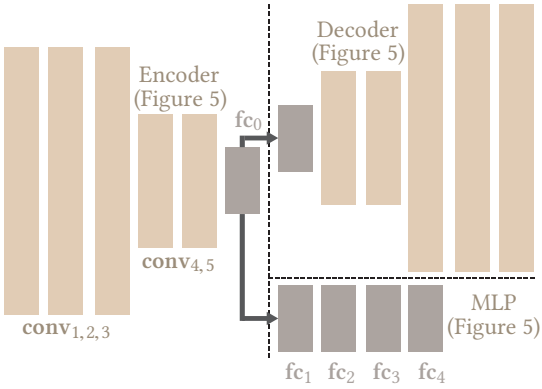

Fig. 6. Our autoencoder uses a mirrored structure (separated by dashed line) with the following specifications: $\mathbf{conv}_1(7, 7, 64, 2)$, $\mathbf{conv}_2(7, 7, 64, 2)$, $\mathbf{conv}_3(5, 5, 128, 2)$, $\mathbf{conv}_4(5, 5, 128, 1)$, $\mathbf{conv}_5(5, 5, 128, 1)$, and $\mathbf{fc}_0(128)$. Here each convolutional layer is specified by its (kernel width, kernel height, #filters, stride), and each fully connected layer is specified by its (#neurons). Our 5-layer Encoder and Decoder part is a scaled down version of AlexNet [Krizhevsky et al. 2012]. AlexNet was originally used for image classification. Here we apply it to the fluid's velocity field due to its uniform grid structure. After autoencoder training, we concatenate $\mathcal{E}$ and an MLP with the following specifications: $\mathbf{fc}_1(128)$, $\mathbf{fc}_2(64)$, $\mathbf{fc}_3(64)$, and $\mathbf{fc}_4(32)$. We use a similar structure as [Peng et al. 2017] for MLP, which is sufficient for representing complex locomotion tasks.

mirrored structure, as illustrated in Figure 6, and separate parameters $\boldsymbol{\theta}_\mathcal{E}, \boldsymbol{\theta}_\mathcal{D}$. Given a dataset of sampled velocity fields $\{\mathbf{u}_{i=1,\cdots,N}\}$ in Equation 1, $\boldsymbol{\theta}_\mathcal{E}$ can be found by optimizing the following cyclic loss function:

$$\underset{\boldsymbol{\theta}_\mathcal{E}, \boldsymbol{\theta}_\mathcal{D}}{\mathbf{argmin}} \sum_{i=1}^{N} \|\mathcal{D}(\mathcal{E}(\mathbf{u}_i, \boldsymbol{\theta}_\mathcal{E}), \boldsymbol{\theta}_\mathcal{D}) - \mathbf{u}_i\|^2. \tag{3}$$

After the training, our state feature is reduced to $\bar{\mathbf{S}} = \left(\mathcal{E}(\mathbf{u})\ \mathbf{v}\ \omega\ \mathbf{c}\ \mathbf{R}\ \mathbf{I}\right)$ where $|\bar{\mathbf{S}}| < 100$ and we use this reduced state vector throughout the rest of the paper. Note that ρ is excluded from the training data and we validate this formulation in two cases. For smoke control, ρ is used for visualization only and does not influence the dynamics at all. Therefore, ρ can be discarded without a loss of information. For liquid control, ρ represents the free-surface boundary condition and discarding it does cause information loss because velocity information is meaningless outside the liquid region. However, we can preserve this information by setting the velocity to be zero outside the liquid region. In our experiments, after a few iterations of optimizing Equation 3, the autoencoder quickly learns to identify liquid regions using this simple treatment.

4.2 TRPO Policy Optimization

Given the encoded low-dimensional state $\bar{\mathbf{S}}$, we can now optimize Equation 2 in a stochastic manner using the TRPO algorithm [Schulman et al. 2015a]. In order to perform this step, we discard the decoder $\mathcal{D}$ after optimizing $\boldsymbol{\theta}_\mathcal{E}$ and concatenate $\mathbf{fc}_0$ in $\mathcal{E}$ with a 4-layer MLP illustrated in Figure 6 parameterized by $\boldsymbol{\theta}_{\mathbf{MLP}}$. TRPO algorithm simultaneously optimizes the $\boldsymbol{\theta} = \left(\boldsymbol{\theta}_\mathcal{E}\ \boldsymbol{\theta}_{\mathbf{MLP}}\right)$, i.e. end-to-end training.

TRPO algorithm is a kind of policy gradient method [Kakade 2001] that limits the search step size by using additional KL divergence constraints. The policy gradient method updates $\boldsymbol{\theta}$ in the direction

Algorithm 1 Training Coupled Fluid/Rigid Controller

```
Initialize θ_0
for i = 0, 1, 2, ... K do
    Sample a set of T trajectories {S̄_0, a_0, S̄_1, ···}
      each having N timesteps
    Update θ_E, θ_D by solving Equation 3 using 10000
      iterations of Adam optimizer [Kingma and Ba 2014]
    Update θ by solving Equation 4
      under the constraint KL(π(θ)|π(θ̄)) < ε
    Update ε dynamically according to
      Algorithm 4.1 in [Nocedal and Wright 2006]
end for
```

of increasing the expected reward $\mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$. [Schulman et al. 2015a] further proved that by optimizing a computationally tractable surrogate reward, the true reward will also reduce at every iteration, thus greatly improving the robustness of the policy gradient method. Specifically, if we update $\boldsymbol{\theta}$ to some arbitrary $\bar{\boldsymbol{\theta}}$, the change in the expected reward is approximated as:

$$\begin{aligned} & \mathbb{E}_{\bar{\boldsymbol{\theta}}}\left[\sum_{i=0}^{\infty} \gamma^i r_i\right] - \mathbb{E}_{\boldsymbol{\theta}}\left[\sum_{i=0}^{\infty} \gamma^i r_i\right] \\ = & \sum_{i=0}^{\infty} \mathbb{E}_{\bar{\mathbf{S}}_i \sim \pi(\bar{\boldsymbol{\theta}})}\left[\gamma^i \mathbb{E}_{\pi(\mathbf{f}_i|\bar{\mathbf{S}}_i, \bar{\boldsymbol{\theta}})}\left[A_{\pi(\bar{\boldsymbol{\theta}})}(\bar{\mathbf{S}}_i, \mathbf{a}_i)\right]\right] \\ \approx & \sum_{i=0}^{\infty} \mathbb{E}_{\bar{\mathbf{S}}_i \sim \pi(\boldsymbol{\theta})}\left[\gamma^i \mathbb{E}_{\pi(\mathbf{f}_i|\bar{\mathbf{S}}_i, \bar{\boldsymbol{\theta}})}\left[A_{\pi(\bar{\boldsymbol{\theta}})}(\bar{\mathbf{S}}_i, \mathbf{a}_i)\right]\right], \end{aligned} \tag{4}$$

where the last equation above is the surrogate reward, where the first expectation is sampled according to current policy $\pi(\boldsymbol{\theta})$ instead of the unknown $\pi(\bar{\boldsymbol{\theta}})$. Here A is the advantage function defined as:

$$A_\pi(\bar{\mathbf{S}}, \mathbf{f}) = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i r_i | \bar{\mathbf{S}}_0 = \bar{\mathbf{S}}, \mathbf{a}_0 = \mathbf{a}\right] - \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i r_i | \bar{\mathbf{S}}_0 = \bar{\mathbf{S}}\right],$$

which measures how much taking a specific action $\mathbf{f}$ will improve the follow-up cumulative reward. In order to have the surrogate reward approximate the true reward, [Schulman et al. 2015a] introduced an additional trust region constraint: $\mathbf{KL}(\pi(\boldsymbol{\theta})|\pi(\bar{\boldsymbol{\theta}})) < \epsilon$, where the difference between $\pi(\boldsymbol{\theta})$ and $\pi(\bar{\boldsymbol{\theta}})$, defined by the Kullback-Leibler divergence $\mathbf{KL}$, must to be smaller than ϵ. Optimizing Equation 4 will decrease the true reward in Equation 4 if ϵ is infinitesimal and the expectations in Equation 4 are evaluated exactly. In practice, the two expectations in Equation 4 are approximated using importance sampling and the trust region size ϵ is tuned dynamically using the standard algorithm in [Nocedal and Wright 2006] to improve the convergence speed of the controller learning algorithm.

4.3 Overall Algorithm

Our final algorithm for learning the coupled system controller interleaves autoencoder learning and policy optimization. In each step, we first sample a set of trajectories $\{\bar{\mathbf{S}}_0, \mathbf{a}_0, \bar{\mathbf{S}}_1, \cdots\}$ according to current $\pi(\boldsymbol{\theta})$ by calling the fluid simulator (Equation 1). Next, we optimize $\boldsymbol{\theta}_\mathcal{E}$ by solving Equation 3. We also update $\boldsymbol{\theta}$ by maximizing Equation 4 under the trust region constraint. And finally we dynamically update the trust region ϵ. The outline of our pipeline is summarized in Algorithm 1.

5 EXPERIMENTS

We evaluate the effectiveness and robustness of our method using a set of benchmarks. Our testbed is a cluster of nodes, each with four 2.5GHz Intel Xeon E5 CPUs (32 cores in all) and 1 GPU (Nvidia GTX1080). A typical 2D controller training takes 10 hours. After the training, 2D controlled liquid or smoke animations can be generated in realtime at 5FPS. In the following, we detail and analyze all the benchmarks.

5.1 Rigid Body Balancing

In our first benchmark, our goal is to have a rigid body balanced at a fixed position. Since the gravity force is applied to the rigid body, a smoke or liquid jet must continuously shoot fluid towards the rigid body to counteract that gravity, achieving dynamic stability. A screenshot of this benchmark is given in Figure 2 (a). We assume that we only have a single fluid jet located at the bottom of Ω_f. We also assume that the fluid jet can move horizontally but not vertically, that the jet's heading can be changed, and finally that the controller can determine whether the jet shoots liquid or not. Therefore, the example-dependent information $\mathbf{I}$ in this benchmark is $\mathbf{I} = \left(x_{\text{jet}}\ \dot{x}_{\text{jet}}\ \beta_{\text{jet}}\ \dot{\beta}_{\text{jet}}\right)^T$ and the action is $\mathbf{a} = \left(\ddot{x}_{\text{jet}}\ \ddot{\beta}_{\text{jet}}\ \delta_{\text{jet}}\right)^T$, where x_{jet} is the fluid jet's horizontal position, β_{jet} is the fluid jet's heading, and finally δ_{jet} is an indicator of whether the jet shoots fluid or not. Note that we use acceleration to control the position and orientation of the fluid jet so that it can move smoothly even if $\pi(\mathbf{a}|\mathbf{S})$ is noisy. The coupled system updates itself according to the following equations:

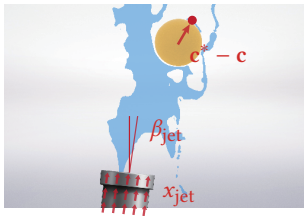

Fig. 7. Illustration of rigid body balancing scenario.

$$\mathcal{F}_{\mathbf{I}}(\mathbf{I},\mathbf{a}) = \begin{pmatrix} \mathbf{min}(\mathbf{max}(x_{\text{jet}} + \dot{x}_{\text{jet}}\Delta t + \ddot{x}_{\text{jet}}\Delta t^2, 0), W) \\ \dot{x}_{\text{jet}} + \ddot{x}_{\text{jet}}\Delta t \\ \mathbf{min}(\mathbf{max}(\beta_{\text{jet}} + \dot{\beta}_{\text{jet}}\Delta t + \ddot{\beta}_{\text{jet}}\Delta t^2, -\beta_{\max}), \beta_{\max}) \\ \dot{\beta}_{\text{jet}} + \ddot{\beta}_{\text{jet}}\Delta t \end{pmatrix}$$

$$\mathbf{f}(\mathbf{I}) = \mathbf{wind}(x_{\text{jet}}, r_{\text{jet}}, \beta_{\text{jet}})\delta_{\text{jet}}.$$

Here r_{jet} is the spout radius of the fluid jet and $\mathbf{wind}(\bullet,\bullet,\bullet)$ is the wind force template centered at x_{jet} as illustrated in Figure 7 as red arrows. $\mathcal{F}_{\mathbf{I}}$ together with Equation 1 completes our definition of $\mathcal{F}$ for the first benchmark.

We evaluated four variants of the rigid body balancing benchmark using two different fluid types, liquid and smoke, and two different rigid body shapes, a ball and a cross. In all four cases we use the following reward function:

$$r(\mathbf{S},\mathbf{a}) = w_{\mathbf{c}}\mathbf{exp}(-\|\mathbf{c}-\mathbf{c}^*\|^2) + w_{\mathbf{v}}\mathbf{exp}(-\|\mathbf{v}\|^2) + w_e(1-\delta_{\text{jet}}), \quad (5)$$

where the first term penalizes position mismatch against the target position $\mathbf{c}^*$, the second term penalizes velocity mismatch, and the last term encourages the controller to use a minimal amount of control forces (indicated by δ_{jet}). For the cross-shaped rigid body, we can control its orientation in addition to its center of mass by adding the following reward function:

$$w_{\dot{\theta}}\mathbf{exp}(-\|\dot{\theta}-\dot{\theta}^*\|^2),$$

where we penalize mismatch in the rotational speed. All the parameters used in these benchmarks are summarized in Table 2.

5.2 Ball Game

In our second benchmark, we use two fluid jets that are independently controlled by two neural-nets with same structure and shared weights. Our example-dependent information $\mathbf{I}$, the action $\mathbf{a}$, and the system updating equation $\mathcal{F}_{\mathbf{I}}$ in this benchmark takes the same form as those in Section 5.1, but we use two separate sets of $\mathbf{I}$, $\mathbf{a}$ for the two jets. These two actions are generated by evaluating the same neural-net twice using mirrored features.

The two jets are supposed to push a rigid body to the other side in order to gain a reward, as illustrated in Figure 2 (c,d). Two different variants of reward functions corresponding to competitive and cooperative actions are discussed below. In addition, we assume that each of these two controllers is unaware of the other controller's action when it tries to optimize the outcome. However, indirect observation is still possible by observing the fluid's velocity field in the scene. This benchmark is more challenging because it is an multi-agent problem and also because this is a partially-observed MDP (POMDP) problem, where each agent does not have complete information about the environment (the other controller's action is unknown). A similar setting is adopted in [Lowe et al. 2017].

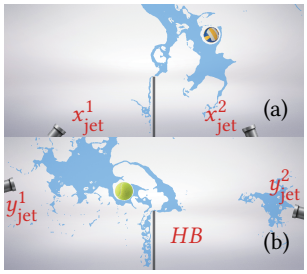

Fig. 8. Illustration of parameters in the two ball game: win-win game (a) and zero-sum game (b).

We use a ballgame-like scene setting for this benchmark. The scene is separated into two halves by a vertical guard-board. The two water jets are located to the left and right of the wall, respectively. As illustrated in Figure 8, we tested two variants of the ballgame.

In our first variant, the two water jets can move freely on the bottom of their half-domain, as two players move as in a volleyball game. Moreover, in this example, the goal of both controllers is to cooperate with each other to push the ball back and forth. This results in a win-win game. To achieve this behavior, we use the following reward function:

$$r_{\text{left}}(\mathbf{S},\mathbf{a}) = w_{\text{side}}\mathbf{exp}(|\mathbf{e}_0^T\mathbf{v}|) + w_{\text{hit}}\delta_{\text{hit}} + w_e(1-\delta_{\text{jet}}) \quad (6)$$

$$r_{\text{right}}(\mathbf{S},\mathbf{a}) = w_{\text{side}}\mathbf{exp}(|\mathbf{e}_0^T\mathbf{v}|) + w_{\text{hit}}\delta_{\text{hit}} + w_e(1-\delta_{\text{jet}}),$$

where we use the first term to determine whether the ball is flying across the guard-board with a high horizontal speed, in which case we reward both controllers. We use δ_{hit} as an indicator of whether the rigid ball hits the ground or the guard-board, in which case we apply a negative reward on both sides. We again use the last term to encourage the controllers to use the minimal amount of control forces.

In the second variant, the two water jets can move freely on the two opposite sides of the simulation domain, in a similar manner as two players standing on the two opposite sides of a pingpong game board. The goal of each controller is to defeat the other controller according to the rules of ball game, i.e. one loses if the rigid ball touches the ground, side wall, or the guard-board on its own side.

This results in a zero-sum game. The ball will bounce back with mirrored vertical velocity when it hits the ceiling. We use the following reward function:

$$r(\mathbf{S}, \mathbf{a}) = \delta_{\text{win}} - \delta_{\text{lose}}, \tag{7}$$

where we use $\delta_{\text{win}}, \delta_{\text{lose}}$ to indicate winner and loser, respectively. Unlike our first variant, these reward functions have no other parameters or additional regularization terms and the sum of reward functions is always zero. The parameters used in both experiments are summarized in Table 3 (parameters with same values as those in Table 2 are omitted).

5.3 Music Ball Player

In our last and most challenging benchmark, we steer the rigid body to accomplish multiple tasks using the same neural-net controller. Our tasks can vary in two ways: First, we steer the rigid body to reach different target positions. In addition, we specify a different expected time of arrival (target time) for the rigid body. By combining these two varieties, we can have a rigid ball to play a small sheet of music through by hitting different pads at certain rhythms. Therefore, the example-dependent information $\mathbf{I}$ as well as the system updating equation are the same as those in Section 5.1, as illustrated in Figure 9.

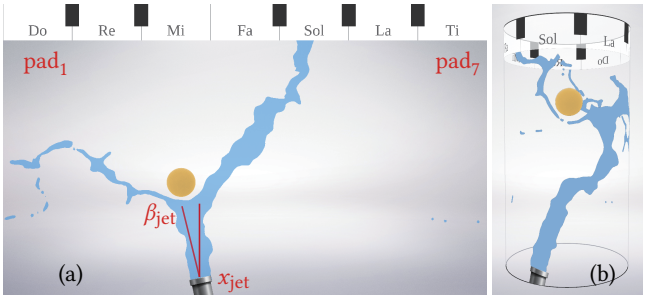

Fig. 9. (a): Illustration of parameters in music ball player. (b): We use periodic boundary condition so that the simulation domain Ω_f can be identified with the side of a cylinder.

We have 7 pads on the ceiling of Ω_f as possible hitting targets. These pads correspond to the 7 notes (Do Re Mi Fa Sol La Ti). To enable multi-task training, we introduce two variables: the target pad id $l^* \in \{1, \cdots, 7\}$ and the target time t^*. Both l^*, t^* are input to the neural-net as additional features in the last layer of encoder $\mathcal{E}$ ($\mathbf{fc}_0$ in Figure 6). We also introduce an additional parameter t_{ac}, which is the accepted time window. Whenever the rigid body hits the pad within $[t^* - t_{ac}, t^* + t_{ac}]$, we consider the hit successful. Finally, we use the following reward function for training:

$$r(\mathbf{S}_i, \mathbf{a}_i) = \begin{cases} w_{\text{fall}} & \delta_{\text{hit}} = 1 \vee |i\Delta t - t^*| \geq t_{ac} \\ w_{\text{err}} & \text{otherwise} \\ w_{\text{hit}} - w_{\text{t}}|i\Delta t - t^*| & l = l^* \wedge |i\Delta t - t^*| < t_{ac} \end{cases}, \tag{8}$$

where we apply a negative reward w_{hit} if the ball falls to the bottom of domain or the hitting time is outside the time window. We apply a negative reward w_{err} if the ball hits a wrong pad. Finally, we reward the controller by w_{hit} when it hits a correct pad, and we use an additional w_{t} term to encourage timely hits. During training, we choose a series of target music pads to be hit. This is done by repeatedly picking a number from $1 - 7$ at random, which is different from the previous number. For each pad ID, we then pick a specific target time to be hit. This is done during the first iteration of Algorithm 1. Following the principle of Common Random Number [Glasserman and Yao 1992], we fix the target positions and the target hitting times for the rest of the iterations. This formulation reduces the variance of importance sampling and allow us to use a smaller number of samples N. In addition, we use a periodic boundary condition along the x-axis. This is because we found it very challenging for the liquid jet to steer the ball to the far right, when it is currently hitting the leftmost pad. As a result, our simulation domain becomes the side of a cylinder. All the parameters in this training are summarized in Table 4 (parameters with the same values as those in Table 2 are omitted).

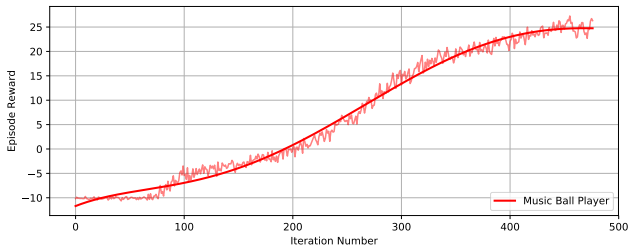

Fig. 10. The average reward plotted against the number of iterations of Algorithm 1 for the music ball player benchmark.

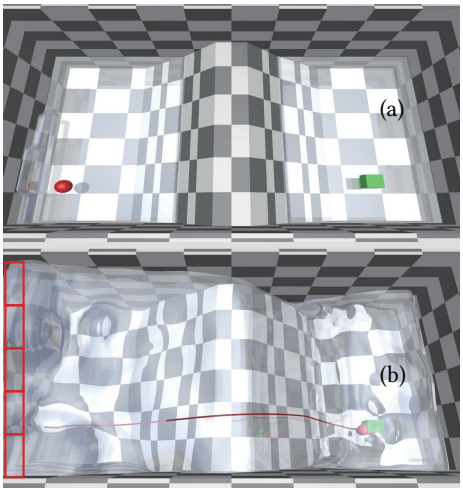

Fig. 11. (a): We control a floating, rigid red ball on the left to cross a high hill in the middle and reach the green target on the right. The controller can modify the water depths at a column of 5 source regions on the left (red boxes). (b): The controlled trajectory of the rigid ball shown as the red curve.

5.4 Controlling the 3D Shallow Water Equation

Extending our method to 3D scenarios is computationally impractical due to the high cost of 3D Navier-Stokes simulation and the large number of simulated samples required by DRL training. In our simulator, for example, collecting 500 samples at a grid-resolution of $160 \times 96 \times 96$ takes approximately 2 hours, so that a full training would require 166 days. However, if we use a simplified fluid model that allows fast simulation such as the shallow water equation [Layton and van de Panne 2002] or the wave equation, then extensions to 3D scenarios are possible.

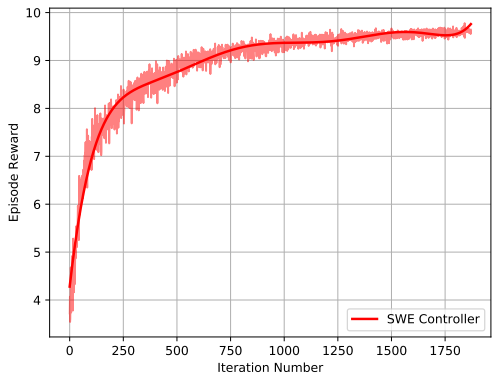

Fig. 12. The average reward plotted against the number of iterations of Algorithm 1 for the 3D shallow water equation benchmark.

In this benchmark, we control a rigid ball floating on 3D water simulated using the shallow water equation, as illustrated in Figure 11. The shallow water equation is discretized and simulated using the numerical scheme proposed in [Kurganov and Petrova 2007], which has proved stability and supports modelling of dry-wet region changes. The 3D scenario is illustrated in Figure 11 (a), which involves two regions separated by a high hill in the middle. The floating ball starts on the left and the goal of the control is to drive it to reach a set of target positions on the right. As a result, the rigid ball must move in all 3 dimensions in order to reach the target. Our controller controls the movement of the ball by changing the depth of the water at a column of 5 source regions on the left boundary. We use the following reward function:

$$r(\mathbf{S}, \mathbf{a}) = \frac{\delta^y_{cross}}{|x - x^*| + 0.1}, \tag{9}$$

where δ^y_{cross} indicates whether the rigid body moves past the target position along the y-axis. When the rigid body passes the target position, we grant a reward based on the distance between the rigid body and the target position along the x-axis. As in the music ball player, our shallow water controller is a multi-task controller because the target position (x^*, y^*, z^*) is taken as a 3-DOF input feature to our neural network. During the training, we sample 48 trajectories in each iteration of Algorithm 1. The target positions for these 48 trajectories are selected at random during the first iteration of Algorithm 1 and fixed in later iterations. This training takes 20 hours and the convergence history is illustrated in Figure 12. All the parameters used in this benchmark are listed in Table 5.

5.5 Analysis

In this section, we analyze the performance of our algorithm and analyze the performance based on four characteristics: computational cost of training, convergence of algorithm, parameter selection, and reward function selection. The correctness and performance of our algorithm can be characterized by the following two properties:

• *Controlled fluid animation generated using our method follows physical laws exactly where* $\mathbf{f}$ *is zero.*

This property follows from Equation 1 and the fact that $\mathbf{f}$ is non-zero in a very small region on the simulation domain boundary. During the simulations, the control can be achieved by injecting liquids into the coupled system instead of applying $\mathbf{f}$. Similarly, this happens only in the boundary region. For the simulator, the influence from the controller can be interpreted as resetting or specifying new boundary conditions, including force, velocity and non-zero divergence, etc. during each simulation step. The property highlighted in this property also explains why our approach can generate controlled fluid animation with better visual plausibility than methods that use non-zero ghost forces throughout the simulation domain such as [Fattal and Lischinski 2004; Shi and Yu 2005].

• *The average reward of a trajectory generated using* π *will reach local minima after Algorithm 1 converges, if* $\epsilon \to 0$ *and* $T \times N \to \infty$. *T is the number of sampled trajectories and N is the per-trajectory length in Line 2 of Algorithm 1.*

This property is shown in [Schulman et al. 2015a] and we apply it to Algorithm 1. Although DRL training is sensitive to the underlying parameters, this property guides users to perform meta-parameter search for $T \times N$ and to design strategies to tune ϵ. Ideally, each iteration of Algorithm 1 should strictly decrease the expected reward. In practice, this depends on whether ϵ is small enough and whether $T \times N$ is large enough. First, if $T \times N$ is large enough then the expected reward will strictly decrease, since we use an adaptive ϵ to guarantee this property. As a result, the only reason for non-decreasing expected reward is a too small N. In this case, the number of sampled trajectories should be increased.

Next, we analyze the performance governed by four characteristics.

5.6 Cost of Training

As mentioned in Section 4, the training cost depends on the number of sampled trajectories, the number of iterations, and the resolution of the simulation grid. For these parameters, we use the values in Table 2 for all our benchmarks. Under these settings, our coupled fluid/rigid system has 50000 − 600000 DOFs and each iteration of the training algorithm takes ∼ 1(min) for liquid control examples and ∼ 5(min) for smoke control examples. This is because a liquid body only occupies a small fraction of simulation domain Ω_f on average, while a smoke body always occupies the entire Ω_f. As a result, it takes more computation for the fluid simulator to solve for the state of smoke body during next timestep. The simulation time for all our examples are summarized in Table 1.

Table 1. Training time for each benchmark until convergence.

Example	Training Time	Example	Training Time
Rigid Body Balancing Liquid+Ball	2.5h	Rigid Body Balancing Smoke+Ball	6.5h
Rigid Body Balancing Liquid+Cross	3h	Rigid Body Balancing Smoke+Cross	11.5h
Cooperative Ball Game	12h	Competitive Ball Game	3h
Music Ball Player	16h	3D Shallow Water Equation	20h

5.7 Convergence of Our Algorithm

In order to evaluate the controller trained by our algorithm, we plot the average reward after each iteration of Algorithm 1. During

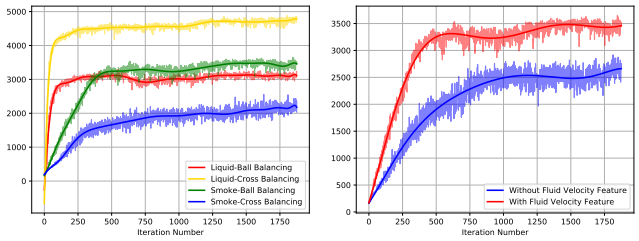

Fig. 13. (a): The average reward plotted against the number of iterations of Algorithm 1 for four variants of the rigid body balancing benchmark: liquid+ball, smoke+ball, liquid+cross, and smoke+cross. Since the rewards are locally noisy, we also fit a solid curve to reflect the time averaged trend of reward. Our algorithm converges within 500 iterations for all 4 variants. (b): We run the smoke+ball setting using our controller and a controller without fluid velocity features, where the output of $\mathcal{E}$ is removed from $\mathbf{fc}_0$. By adding a fluid velocity feature, our controller converges a lot faster and the controller achieves a higher reward on convergence.

training, less failure cases happen as the algorithm converges. The convergence history of rigid ball balancing benchmark (Section 5.1) is shown in Figure 13 (a). Our algorithm usually converges within 500 iterations, which involve samples collected from 1.7 million fluid simulations. In addition, we compared our encoder+MLP controller with an MLP-only controller in Figure 13 (b). It shows that by adding fluid velocity features, our algorithm converges faster and the controller achieves a significantly higher average reward on convergence. To achieve such higher performance, we found it important to update both the autoencoder and the MLP in each iteration of Algorithm 1. In fact, the autoencoder is updated in two places (Line 5 and Line 7) in our current implementation. According to our experiments, one can choose to only update $\boldsymbol{\theta}_{\mathbf{MLP}}$ in Line 7 without a loss of performance. For the more challenging cooperative ball game benchmark (Section 5.2), the training converges faster than the simpler rigid body balancing benchmarks and the learned control policy looks very effective in pushing the ball to the other side. The convergence history of this variant is illustrated in Figure 14. However, for the competitive ball game benchmark, we observe that a ball game between two highly skilled player should last for a long period of time. Therefore, we plot the average number of timesteps before the game ends (in Figure 15). The shape of this plot is consistent with our observation where the average game length increases and eventually converges. Finally the convergence

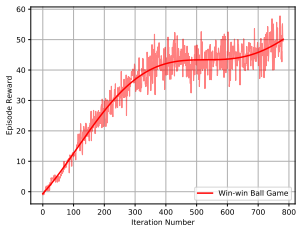

Fig. 14. The average reward plotted against the number of iterations of Algorithm 1 for the win-win ball game example.

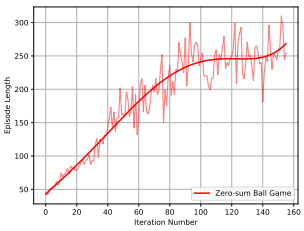

Fig. 15. The average timesteps before game ends plotted against the number of iterations of Algorithm 1 for the zero-sum ball game example.

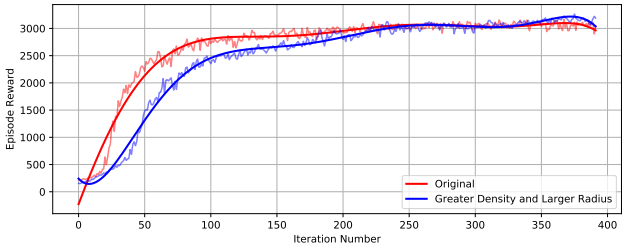

Fig. 16. A comparison of two convergence histories of liquid+ball benchmark. In the second example (blue), we twiced the rigid ball radius, we also twiced rigid body density. The controller performs equally well on convergence on the rescaled reward function in Table 2.

history for the music ball player benchmark (Section 5.3) is provided in Figure 10. Under this setting, the trained controller performs very accurately after 500 iterations.

5.8 Parameter Selection

The performance of trained controller using our method can be sensitive to the parameters. There are two sets of parameters used in our benchmarks. The first set of parameters determine the scenarios, including size of Ω_f, max tilt angle β_{jet}, or the density of rigid body. Currently we set these parameters for each benchmark, as explained in Section 5.1, 5.2, and 5.3. The second set of parameters are the weights of the reward functions. We choose these parameters using grid-based meta-parameter search, which is well known and widely used in machine learning community to avoid training falling into local minima (see e.g., [Claesen and De Moor 2015] for an introduction). Specifically, for a set of reward weights, we fix one of them to 1 and run multiple DRL training using different values for other weights. The grid size for search is always 1 except for w_e in Table 2 for which we use 0.01. After meta-parameter search, we use the controller that performs the best on a same rescaled reward. The weights we show in Table 2, Table 3, and Figure 10 are the weights of the rescaled reward. Meta-parameter search makes our method more friendly to end-user and less sensitive to manual tweaking. Although we can also take the first set of parameters into the search, we found that our results are less sensitive to them. For example, in Figure 16 we show another rigid ball balancing example with a different rigid ball radius, gravitational coefficient, and rigid body density. After meta-parameter search, our controller performs equally well on the rescaled reward function Equation 5 with weights in Table 2.

5.9 Reward Function Selection

One major issue in using DRL is that the selection of reward function can be arbitrary and sometimes requires manual tweaking. Most of the reward functions used in our benchmarks (Equation 5, Equation 6, and Equation 7) combine exponential terms and binary terms. These terms and functions are chosen to be similar to those in character control [Liu and Hodgins 2017; Peng et al. 2017]. However, the reward function used by our music ball player (Equation 8) has a non-standard form. Parameters in the reward functions are computed using grid-based meta-parameter search.

6 CONCLUSION, LIMITATIONS AND FUTURE WORK

We present a learning-based control approach for coupled fluid/rigid systems. In our approach, we use DRL to compute the ghost forces the boundary of the simulation domain to control the system. As a result, the natural internal dynamics of the fluid and rigid body are preserved. Our controller can automatically handle various controlling tasks involving high-dimensional state features, different control objectives, and a different number of tasks. By designing an auto-encoder-facilitated DRL neural-net and simultaneously training the autoencoder and control policy, our approach achieves a high convergence rate. Control decisions through neural-nets after training can be generated in real-time during runtime simulation on a desktop machine.

Our method has some inherent limitations. For example, our current algorithm is sensitive to the specific implementation of the fluid solver, the scenario and reward function, and usually requires re-training if these parameters change. This implies that transferring the learned control skills to new fluid simulators or real-life scenarios is difficult. In addition, our method inherits the limitations of a general deep learning algorithm. The method does not guarantee successful control for arbitrary tasks. For example, using only non-loop boundary in the music ball play benchmark may not result in satisfactory results, mainly because the limited controls from a single jet do not support timely hit on targets.

In the future, we would like to try more challenging control problems, e.g., scenarios involving heavier rigid bodies and moving music pads. We also plan to extend our framework to handle multiple interacting rigid bodies, where the control objectives can be formulated based on their interactions. Due to the generality of our algorithm, we are also considering extending our method to control a 3D Navier-Stokes system. To alleviate the prohibitive computational overhead in training, recent research in improving the sampling efficiency of DRL such as [Mnih et al. 2016; Schulman et al. 2015b, 2017] can be used. Another way to reduce the computational overhead of the 3D simulator is by using GPU acceleration as is done in [Chentanez and Müller 2011].

ACKNOWLEDGMENTS

This paper is supported by the National Key R&D Program of China (2017YFB1002701) and Natural Science Foundation of China (61602265). Zherong Pan and Dinesh Manocha are supported in part by ARO Contract (W911NF-14-1-0437) and NSF award (1305286). Finally, we thank Prof. Jin Huang and Prof. Changxi Zheng for useful discussions.

REFERENCES

Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. 2012. Versatile Rigid-fluid Coupling for Incompressible SPH. *ACM Trans. Graph.* 31, 4, Article 62 (July 2012), 8 pages.

Christopher Batty, Florence Bertails, and Robert Bridson. 2007. A Fast Variational Framework for Accurate Solid-fluid Coupling. *ACM Trans. Graph.* 26, 3, Article 100 (July 2007).

Markus Becker and Matthias Teschner. 2007. Weakly Compressible SPH for Free Surface Flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '07)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 209–217.

Markus Becker, Hendrik Tessendorf, and Matthias Teschner. 2009. Direct Forcing for Lagrangian Rigid-Fluid Coupling. *IEEE Transactions on Visualization and Computer Graphics* 15, 3 (May 2009), 493–503.

Boris Bonev, Lukas Prantl, and Nils Thuerey. 2017. Pre-computed Liquid Spaces with Generative Neural Networks. *arXiv* to appear (Apr 2017), 14.

Nuttapong Chentanez and Matthias Müller. 2011. Real-time Eulerian Water Simulation Using a Restricted Tall Cell Grid. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 82, 10 pages. https://doi.org/10.1145/1964921.1964977

Marc Claesen and Bart De Moor. 2015. Hyperparameter Search in Machine Learning. *arXiv preprint arXiv:1502.02127* (2015).

Douglas Enright, Stephen Marschner, and Ronald Fedkiw. 2002. Animation and Rendering of Complex Water Surfaces. *ACM Trans. Graph.* 21, 3 (July 2002), 736–744.

Isabelle Fantoni and Rogelio Lozano. 2001. *Non-Linear Control for Underactuated Mechanical Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Raanan Fattal and Dani Lischinski. 2004. Target-driven Smoke Animation. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 441–448.

Paul Glasserman and David D. Yao. 1992. Some Guidelines and Guarantees for Common Random Numbers. *Manage. Sci.* 38, 6 (June 1992), 884–908.

Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evol. Comput.* 9, 2 (June 2001), 159–195.

Sham Kakade. 2001. A Natural Policy Gradient. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani (Eds.). MIT Press, 1531–1538.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. Curran Associates Inc., USA, 1097–1105.

Alexander Kurganov and Guergana Petrova. 2007. A Second-Order Well-Balanced Positivity Preserving Central-Upwind Scheme for the Saint-Venant System. *Commun. Math. Sci.* 5, 1 (03 2007), 133–160. https://projecteuclid.org:443/euclid.cms/1175797625

Anita T. Layton and Michiel van de Panne. 2002. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer* 18, 1 (01 Feb 2002), 41–53. https://doi.org/10.1007/s003710100131

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end Training of Deep Visuomotor Policies. *J. Mach. Learn. Res.* 17, 1 (Jan. 2016), 1334–1373.

Libin Liu and Jessica Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Trans. Graph.* 36, 3, Article 29 (June 2017), 14 pages.

Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. 2008. Two-Way Coupled SPH and Particle Level Set Fluid Simulation. *IEEE Transactions on Visualization and Computer Graphics* 14, 4 (July 2008), 797–804.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. 6382–6393.

Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. 2011. *Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction*. Springer Berlin Heidelberg, Berlin, Heidelberg, 52–59.

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 449–456.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 1928–1937.

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. 2009. Energy-preserving Integrators for Fluid Animation. *ACM Trans. Graph.* 28, 3, Article 38 (July 2009), 8 pages.

J. Nocedal and S. J. Wright. 2006. *Numerical Optimization* (2nd ed.). Springer, New York.

Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. 2013. Interactive Localized Liquid Motion Editing. *ACM Trans. Graph.* 32, 6, Article 184 (Nov. 2013), 10 pages.

Zherong Pan and Dinesh Manocha. 2017. Efficient Solver for Spacetime Control of Smoke. *ACM Trans. Graph.* 36, 5, Article 162 (July 2017), 13 pages.

Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Trans. Graph.* 36, 4, Article 41 (July 2017), 13 pages.

Jovan Popović, Steven M. Seitz, and Michael Erdmann. 2003. Motion Sketching for Control of Rigid-body Simulations. *ACM Trans. Graph.* 22, 4 (Oct. 2003), 1034–1054.

Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. 2000. Interactive Manipulation of Rigid Body Simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 209–217.

Karthik Raveendran, Chris Wojtan, Nils Thuerey, and Greg Turk. 2014. Blending Liquids. *ACM Trans. Graph.* 33, 4, Article 137 (July 2014), 10 pages.

Avi Robinson-Mosher, Tamar Shinar, Jon Gretarsson, Jonathan Su, and Ronald Fedkiw. 2008. Two-way Coupling of Fluids to Rigid and Deformable Solids and Shells. In *ACM SIGGRAPH 2008 Papers (SIGGRAPH '08)*. ACM, New York, NY, USA, Article 46, 9 pages.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015a. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 1889–1897.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015b. High-Dimensional Continuous Control Using Generalized Advantage Estimation. (2015). arXiv:arXiv:1506.02438

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. (2017). arXiv:arXiv:1707.06347

Lin Shi and Yizhou Yu. 2005. Taming Liquids for Rapidly Changing Targets. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '05)*. ACM, New York, NY, USA, 229–236.

Nils Thuerey. 2016. Interpolations of Smoke and Liquid Simulations. *ACM Trans. Graph.* 36, 1, Article 3 (Sept. 2016), 16 pages.

J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. 2016. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *ArXiv e-prints* (July 2016). arXiv:cs.CV/1607.03597

Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. 2003. Keyframe Control of Smoke Simulations. *ACM Trans. Graph.* 22, 3 (July 2003), 716–723.

Christopher D. Twigg and Doug L. James. 2007. Many-worlds Browsing for Control of Multibody Dynamics. *ACM Trans. Graph.* 26, 3, Article 14 (July 2007).

Christopher D. Twigg and Doug L. James. 2008. Backward Steps in Rigid Body Simulation. *ACM Trans. Graph.* 27, 3, Article 25 (Aug. 2008), 10 pages.

Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. 2012. Optimizing Locomotion Controllers Using Biologically-based Actuators and Objectives. *ACM Trans. Graph.* 31, 4, Article 25 (July 2012), 11 pages.

Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. 2017. How to Train Your Dragon: Example-guided Control of Flapping Flight. *ACM Trans. Graph.* 36, 6, Article 198 (Nov. 2017), 13 pages.

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. ACM, New York, NY, USA, Article 105.

Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. 2010. *Deconvolutional networks.* 2528–2535.

Bo Zhu, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. 2013. A New Grid Structure for Domain Extension. *ACM Trans. Graph.* 32, 4, Article 63 (July 2013), 12 pages.

Yongning Zhu and Robert Bridson. 2005. Animating Sand As a Fluid. *ACM Trans. Graph.* 24, 3 (July 2005), 965–972.

A PARAMETERS

We list all the parameters used in our benchmarks. For all the position variables such as $\mathbf{c}, \mathbf{c}^*, x_{jet}, y_{jet}$, we use unit (m). For all the velocity variables such as $\mathbf{v}$, we use unit (m/s). For all the orientation variables such as θ, we use unit (rad). For all the angular velocity variables such as $\dot{\theta}$, we use unit (rad/s).

Table 2. Common parameters used in four rigid body balancing benchmarks. Note that the absolute densities of fluid and rigid bodies do not alter dynamics behaviors so that we only show their density ratio.

Parameter	Explanation	Value
W, H	Width,Height of Ω_f	5,3(m)
$x_{\text{jet}} \in [a_{\text{jet}}, b_{\text{jet}}]$	Horizontal range of water jet movement	$[0, W]$
y_{jet}	Vertical position of water jet	0
$\beta_{\max}$	Max tilt angle of water jet	60°
n_W, n_H	Effective number of grid cells along W,H	160,96
K	Max number of iterations in Algorithm 1	2000
T	Number of trajectories collected in each iteration	32
N	Number of timesteps per trajectory	500
r_{jet}	Spout radius	0.12(m)
$\mathbf{c}^*$	Target position	2.5,1.5(m)
$\dot{\theta}^*$	Target rotational speed	±1(rad/s)
$w_{\mathbf{c}}, w_{\dot{\theta}}, w_{\mathbf{v}}, w_e$	Weights of reward	10,5,2,1
d_r/d_f	Density ratio between rigid bod d_r and fluid d_f	1.5
g	Gravity coefficient	$9.81(ms^{-2})$
Δt	Timestep size	0.02(s)

Table 3. Additional parameters used in ball game benchmarks. Other parameters are the same as those in Table 2.

Parameter	Explanation	Value
W, H	Width, Height of Ω_f	5, 3 (m)
Win-Win Parameters	**Explanation**	**Value**
$x^1_{\text{jet}} \in [a^1_{\text{jet}}, b^1_{\text{jet}}]$	Horizontal range of left jet movement	$[0, W/2]$
$x^2_{\text{jet}} \in [a^1_{\text{jet}}, b^2_{\text{jet}}]$	Horizontal range of right jet movement	$[W/2, W]$
$y^{1,2}_{\text{jet}}$	Vertical position of the two jets	0
HB	Height of the guard-board	$W/3$
$w_{\text{side}}, w_{\text{hit}}, w_{\text{e}}$	Weights of reward	$5, -1, 0.05$
Zero-Sum Parameters	**Explanation**	**Value**
$y^1_{\text{jet}} \in [a_{\text{jet}}, b_{\text{jet}}]$	Vertical range of left jet movement	$[0, H]$
$y^2_{\text{jet}} \in [a_{\text{jet}}, b_{\text{jet}}]$	Vertical range of right jet movement	$[0, H]$
$x^1_{\text{jet}}, x^2_{\text{jet}}$	Horizontal position of the two jets	$0, W$
HB	Height of the guard-board	2W/5

Table 4. Parameters used in the music ball player benchmark.

Parameter	Explanation	Value
N	Number of timesteps per trajectory	500
t_{ac}	Accepted time window	1
$w_{\text{fall}}, w_{\text{err}}, w_{\text{hit}}, w_{\text{t}}$	Weights of reward	$-10, -1, 3, 1$
d_r/d_f	Density ratio between rigid bod d_r and fluid d_f	1.2

Table 5. Parameters used in the 3D shallow water equation benchmark.

Parameter	Explanation	Value
W, H	Width, Height of Ω_f	2.5, 1.5 (m)
D	Depth of the water tank	1 (m)
H_{hill}	Height of the hill	0.6 (m)
D_{init}	Initial water depth	0.37 (m)
D_{src}	Maximal water depth at the source region	0.75 (m)
n_W, n_H	Effective number of grid cells along W,H	160,96
T	Number of trajectories collected in each iteration	48
N	Number of timesteps per trajectory	500
d_r/d_f	Density ratio between rigid bod d_r and fluid d_f	0.1